NORTHWESTERN POLYTECHNIC UNIVERSITY

# k-Nearest-Neighbors (k-NN) Algorithm

**Prepared For:**

Mr. Henry Chang
Machine Learning and Business Intelligence CS550
Summer 2021
Northwestern Polytechnic University

**Prepared By:**

Ms. Nagalla, Santhi Sree ID:19568

# Table of Contents

# What is k-Nearest-Neighbors (k-NN) ?

- k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

# Example - Mobile device sensors

Most mobile devices are equipped with different kind of sensors.

We can use the data sent from Gyroscope sensor and Accelerometer sensor to categorize any motion:

- ➢ 3 numbers from Accelerometer sensor.
- ➢ 3 numbers from Gyroscope sensor.

| Accelerometer Data | | | Gyroscope Data | | | Fall (+), Not (-) |
|---|---|---|---|---|---|---|
| x | y | z | x | y | z | +/- |
| 1 | 2 | 3 | 2 | 1 | 3 | - |
| 2 | 1 | 3 | 3 | 1 | 2 | - |
| 1 | 1 | 2 | 3 | 2 | 2 | - |
| 2 | 2 | 3 | 3 | 2 | 1 | - |
| 6 | 5 | 7 | 5 | 6 | 7 | + |
| 5 | 6 | 6 | 6 | 5 | 7 | + |
| 5 | 6 | 7 | 5 | 7 | 6 | + |
| 7 | 6 | 7 | 6 | 5 | 6 | + |
| 7 | 6 | 5 | 5 | 6 | 7 | ?? |

Prediction

# Calculate Euclidean Distance

- Calculate the distance between two vectors using the Euclidean distance measure. It is calculated as the square root of the sum of the squared differences between the two vectors.
    - Euclidean Distance = sqrt(sum i to N (x1_i – x2_i)^2)
- Where x1 is the first row of data, x2 is the second row of data and i is the index to a specific column as we sum across all columns.
- With Euclidean distance, the smaller the value, the more similar two records will be. A value of 0 means that there is no difference between two records.

## Get Nearest Neighbors

- To locate the neighbors for a new piece of data within a dataset we must first calculate the distance between each record in the dataset to the new piece of data.
- Once distances are calculated, we must sort all of the records in the training dataset by their distance to the new data. We can then select the top k to return as the most similar neighbors.

# Get Nearest Neighbors - Formula

- A general rule of thumb: K = the closest odd number of the square root of the number of samples.
  - ➢ K = sqrt(number of data samples)
- If no winner, then pick the next odd number greater than K
- Since K = 3, we select the 3 closest neighbors.

Example = sqrt(number of data samples)

=sqrt(9) = 3

# Make Predictions

- The KNN prediction of the query instance is based on simple majority of the category of nearest neighbors.
  - In our example, the data is only binary, thus the majority can be taken as simple as counting the number of ‘+' and ‘-‘ signs.
  - If the number of plus is greater than minus, we predict the query instance as plus and vice versa.
  - If the number of plus is equal to minus, we can choose arbitrary or determine as one of the plus or minus.

# Find the Value of K?

| x1 | y1 | z1 | x2 | y2 | z2 | FallOrNot | "Distance to each neighbor for Accelerometer & Gyroscope Sensor<br>= (Targetx1-Datax1)^2+(Targety1-Datay1)^2<br>+(Targetz1-Dataz1)^2+ (Targetx2-Datax2)^2<br>+(Targety2-Datay2)^2 +(Targetz2-Dataz2)^2<br>= (7-X1)^2+(6-y1)^2+(5-z1)^2+(5-x2)^2+(6-y2)^2+(7-z2)^2" | "K<br>=Number of nearest neighbors<br>=sqrt(number of neighbors)<br>=sqrt(number of data samples)<br>=sqrt(8)<br>=3" |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 1 | 3 | - | (7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2 = 106 | |
| 2 | 1 | 3 | 3 | 1 | 2 | - | (7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2 = 108 | |
| 1 | 1 | 2 | 3 | 2 | 2 | - | (7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2 = 115 | |
| 2 | 2 | 3 | 3 | 2 | 1 | - | (7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2 = 101 | |
| 6 | 5 | 7 | 5 | 6 | 7 | + | (7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2 = 6 | + |
| 5 | 6 | 6 | 6 | 5 | 7 | + | (7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2 = 7 | + |
| 5 | 6 | 7 | 5 | 7 | 6 | + | (7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2= 10 | |
| 7 | 6 | 7 | 6 | 5 | 6 | + | (7-7)^2+(6-6)^2+(5-5)^2+(5-6)^2+(6-5)^2+(7-6)^2 = 7 | + |
| 7 | 6 | 5 | 5 | 6 | 7 | ? ( + ) | | |

# Program to find the Euclidean Distance

```
# calculate the Euclidean distance
between two vectors
#     Euclidean Distance = sqrt(sum
i to N (x1_i – x2_i)^2)
# Result:
#    10.295630140987
#    10.392304845413264
#    10.723805294763608
#    10.04987562112089
#    2.449489742783178
#    2.6457513110645907
#    3.1622776601683795
#    2.6457513110645907
```

```
def euclidean_distance(row1,
row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] -
row2[i])**2
    return sqrt(distance)
```

# Locate the Neighbors

```
# Locate the most similar neighbors
# Result
#   [6,5,7,5,6,7,1],
#   [5,6,6,6,5,7,1],
#   [7,6,7,6,5,6,1]]
```

```
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

# Classification prediction with Neighbors

```
# Make a classification prediction with neighbors.
# - test_row is row 0
# - num_neighbors is 3
```

```
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

# Predict value on Test Data

```
# Test distance function
dataset =  [[7,6,5,5,6,7,1],
            [1,2,3,2,1,3,0],
            [2,1,3,3,1,2,0],
            [1,1,2,3,2,2,0],
            [2,2,3,3,2,1,0],
            [6,5,7,5,6,7,1],
            [5,6,6,6,5,7,1],
            [5,6,7,5,7,6,1],
            [7,6,7,6,5,6,1]]
```

```
# Calculate euclidean_distance
print("Euclidean distance between two vectors")
for i in range(1,len(dataset)):
    print(euclidean_distance(dataset[0],dataset[i]))
# row 0 (i.e., dataset[0]) is the one to be predicted
prediction = predict_classification(dataset, dataset[0], 3)
# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
#    Expected 1, Got 1.
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

# Pros and Cons of kNN

KNN is very easy to implement. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

KNN not work well in below conditions either manual or program -

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn’t work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

# Run on Colab

https://colab.research.google.com/notebooks/intro.ipynb#recent=true

Knn_Python.ipynb

File Edit View Insert Runtime Tools Help Last saved at 3:24 PM

+ Code + Text

```
              [6,5,7,5,6,7,1],
              [5,6,6,6,5,7,1],
              [5,6,7,5,7,6,1],
              [7,6,7,6,5,6,1]]

# Caluclate euclidean_distance
print("Euclidean distance between two vectors")
for i in range(1,len(dataset)):
  print(euclidean_distance(dataset[0],dataset[i]))

# row 0 (i.e., dataset[0]) is the one to be predicted
prediction = predict_classification(dataset, dataset[0], 3)

# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
#    Expected 1, Got 1.
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

```
Euclidean distance between two vectors
10.295630140987
10.392304845413264
10.723805294763608
10.04987562112089
2.449489742783178
2.6457513110645907
3.1622776601683795
2.6457513110645907
Expected 1, Got 1.
```

# REFERENCES


- https://npu85.npu.edu/~henry/npu/classes/data_science/algorithm/slide/k_nn_example.html
- https://npu85.npu.edu/~henry/npu/classes/data_science/algorithm/slide/knn_from_scratch.html
- Google Drive URL -

  https://docs.google.com/presentation/d/1DitorH-ERPAH_HDlwhXghNG-zT0l_tszva81A-uE2b8/edit?usp=sharing

- GitHub Link - https://github.com/santhinagalla/Machine-Learning/tree/main/Supervised%20Learning/Falling%20Prediction%20using%20KNN